Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. VI (2011), No. 1 (March), pp. 187-195

Using Fixed Priority Pre-emptive Scheduling in Real-Time Systems

D. Zmaranda, G. Gabor, D.E. Popescu, C. Vancea, F. Vancea

Doina Zmaranda, Gianina Gabor, Daniela Elena Popescu
Codruta Vancea, Florin Vancea
University of Oradea
Romania, 410087 Oradea, 1 Universitatii St.
E-mail: {zdoina,gianina,depopescu,cvancea,fvancea}@uoradea.ro

Abstract: For real-time applications, task scheduling is a problem of paramount importance. Several scheduling algorithms were proposed in the literature, starting from static scheduling or cyclic executives which provide very deterministic yet inflexible behaviour, to the so called best-effort scheduling, which facilitates maximum run-time flexibility but allows only probabilistic predictions of run-time performance presenting a non-predictable and non-deterministic solution. Between these two extremes lies fixed priority scheduling algorithms, such as Rate Monotonic, that is not so efficient for real-time purposes but exhibits a predictable approach because scheduling is doing offline and guarantees regarding process deadlines could be obtained using appropriate analysis methods. This paper investigates the use of Rate Monotonic algorithm by making adjustments in order to make it more suitable for real-time applications. The factors that motivate the interest for fixed priority scheduling algorithms such Rate Monotonic when doing with real-time systems lies in its associated analysis that could be oriented in two directions: schedulability analysis and analysis of process interactions. The analyzing process is carried out using a previously implemented framework that allows modelling, simulation and schedulability analysis for a set of real-time system tasks, and some of the results obtained are presented.
Keywords: real-time systems, fixed priority preemptive scheduling.

1 Introduction

Real-time systems are often safety critical and require a high quality design in order to obtain and guarantee the requested properties. The design process consists in building models on which the required system properties are assessed; based on this previously developed models an implementation that preserves these properties is further developed.

In order to develop a large-scale real-time system we must be able to manage both the logical complexity and timing complexity using a highly disciplined approach [10].The problem of dealing with logical complexity is addressed by the several existing software engineering general methodologies [11] while timing complexity represents an issue that is addressed by specific scheduling algorithms.

For real-time systems, two modelling approaches are known in the literature: first of it allows handling of traditional, periodically sampled control systems, and is represented by the so called timed-triggered approach; the second type of model deals with discrete event systems, and it is known as the event-triggered approach. If the main advantage of event-driven approach is flexibility and better resource utilization, the main advantage of time-driven approach is predictability.

In the time-triggered approach, all communication and processing activities are initiated at predetermined points in time: there is only one interrupt and that is the periodic clock interrupt, which partitions the continuum time into sequences of equally spaced granules. Timed-triggered tasks are characterized by a period and a deadline; also, knowledge about task's Worst case Execution Time (WCET) is generally assumed.

On the other side, event-triggered approach is dictated by the external environment: all communication and processing activities are initiated whenever a significant change of state, i.e., an event other than regular event of a clock tick, is noted. The signalling of significant events is realized by the well-known interrupt mechanism. The event-triggered based systems require a scheduling strategy to achieve the appropriate software task that services the event.

In practice, for several real-time applications event-triggered tasks are sporadic, and exhibit a predictable inter-arrival time. Thus, this time could be seen as task period, deadline being smaller or equal with this [15]. In practical situations, when dealing with hard real-time issues, mixed systems are often encountered [13]. For these, a common approach is to model the system as a timed-triggered one, and deal with events as periodic tasks with inter-arrival times considered as their period. Of course, if the system requires handling of urgent events, that implies a pre-emptive scheduling strategy to be able to meet those deadlines.

Consequently, when we are modelling such real-time mixed systems, choosing the right scheduling strategy is an important aspect, and several issues has to be considered [17]: if we are considering time-triggered tasks, a static scheduling proves to be efficient both for scheduling and for the communications purposes. It is important into the design phase to correctly divide system functionality into tasks and further, tasks with long periods should be statically divided into subtasks with shorter periods; if sporadic event-triggered tasks occur and they have shorter deadline that execution time of another task, allowing pre-emption is mandatory and therefore leading to a pre-emptive scheduling approach.

Therefore, using a static and pre-emptive scheduling strategy together with some initial reasonable assumptions when constructing task's model, could provide a solution for analysing and developing mixed systems [8]. In this case, as the static approach, we consider that fixed priority assignment can be adopted without loosing the benefits of the fully static approach.

2 Fixed priority pre-emptive scheduling and real-time systems

Fixed priority scheduling algorithms exhibit a predictable approach: because scheduling is doing offline, guarantees regarding process deadlines could be obtained using appropriate analysis methods [5]. Fixed priority scheduling has been often criticized as being too static by the supporters of best effort scheduling and too dynamic by the supporters of cyclic executives. For building a mixed real-time system from a number of periodic tasks and several sporadic tasks, static priority pre-emptive scheduler implies that at run time the highest priority task is run, this pre-empting other lower priority tasks [6].

From a historical perspective, Rate Monotonic scheduling algorithm is the most appropriate in this sense, because it is pre-emptive and because it is known to be the optimal in their respective classes [14].

Rate Monotonic scheduling algorithm is an example of a priority driven algorithm with static priority assignment [2], in the sense that the priorities of all requests are known before their arrival, the priorities for each task being the same and known a-priori (they are determined only by the period of task).

Therefore, for the time being, Rate Monotonic is used in most practical applications [3]. The reasons for this choice are the following: easy to implement an to analyze; however, the schedulability assessment given by Liu and Layland [12] is sufficient (all task sets that pass the

test are guaranteed to be schedulable) but not necessary (a task set that fails to pass the test is not necessarily unschedulable) [12]; more predictable, especially in high overloaded conditions; this prediction is linked to several initial simplifying assumptions and restrictions that Rate Monotonic has: all tasks are independent and periodic, deadline is equal to their period.

The factors that motivated the interest for fixed priority scheduling algorithms such Rate Monotonic when doing with real-time systems lies in its associated analysis, which could be oriented in two directions [3]: *schedulability analysis* based on worst case execution times of the processes; one property of the early utilization schedulability analysis is its simplicity both in concept and in computational complexity. This simplicity comes from the initial assumptions that are based on constraints upon characteristics of all processes (all processes are periodic and must have deadline equal to their period) and assumption that process priorities given by their period (according to rate monotonic policy) are correctly assigned, otherwise analysis is not efficient and *inclusion of aperiodic processes* by making them periodic based on estimated inter-arrival times.

Rate Monotonic priority assignment policy [12] states that process deadlines must be equal with their respective periods $D_i = T_i$. This assignment could be restrictive, especially for hard real-time sporadic tasks that have deadlines not related to their inter-arrival times, and hence they cannot be modelled as simple periodic tasks with period equal with deadline. For this case the following relation holds:

$$C_i \leq D_i \leq T_i \tag{1}$$

where: C_i represents the computation time for task i D_i represents task i deadline T_i represents task i period

A variation of original Rate Monotonic, called Deadline Monotonic assign priorities in inverse order to the task deadlines. Deadline Monotonic algorithm is equivalent with Rate Monotonic when, for all processes $D_i = T_i$. Deadline Monotonic priority assignment is optimal in a similar manner to Rate Monotonic if there is a feasible priority ordering over a set of processes, a deadline monotonic priority ordering over those processes will be also feasible [4]. Both Rate Monotonic and Deadline Monotonic approaches assume that all processes have a common release time: if processes are permitted to have arbitrary offsets, then optimality could be affected [8,14]. Under these circumstances, neither priority assignment is optimal [9]; but, if controlled offset is allowed this property may be not significant affected.

3 Considering the overheads

Even if in most models overhead induced by scheduling is neglected and considered 0, this is not the case in reality. Generally, implementation scheme for fixed priority schedulers implies maintaining at least two queues: a ready queue and a waiting queue. The ready queue contains the tasks that are ready for execution and the waiting queue contains tasks that have already been executed and they are waiting for the next period. The queues are ordered in different ways: the ready queue is ordered based on task priority (the most priority task first - for Rate Monotonic the task with lower period T_i is the most priority task) and the waiting queue is ordered based on the starting time of the task R_i.

If a task from the waiting queue becomes ready for execution, then it is moved to the ready queue according to its priority; if the priority of the first task from the ready queue becomes bigger than the priority of the current task, then a context switch will occur [4]. This leads to two kinds of overheads: *context switching overhead* - being the time needed to pre-empt a task, save its context and load the context of another task and *scheduling overhead* - the time taken to move newly arrived or pre-empted tasks between the two queues.

The common ways to consider these times into a system model implies adding the overhead times to the known times according to the following [6]: for *context switching times*, the simplest way to do it is to increase task computation time C_i of all tasks with the double of estimated time needed to do of doing the context switch C_{sw} :

$$C_i = C_i + 2 * C_{sw} \tag{2}$$

and *for scheduling overhead* the same approach could be used, this time being added the scheduling overhead time C_{sch} :

$$C_i = C_i + C_{sch} \tag{3}$$

Generally, when considering the total of all overheads, an average could be calculated over all tasks, denoted by C_{ov}, and could be added to each computation time:

$$C_i = C_i + C_{ov} \tag{4}$$

This approach of measuring the total system overhead, averaging it and including in all computation times is simple to be modelled and used [16]. Another way to include overheads into the model could be considered and modelled as additional tasks, but this complicates it too much and sometimes is unnecessarily.

When constructing a real-time system model, from the accuracy point of view, it is important to take into consideration these overheads. But, an important aspect is represented also by the possibilities of reducing the overheads impact on the overall system's performance [20]. Because most of these overheads are linked with pre-emption (both context switching and scheduling overheads occur after a task pre-emption), a possibility of doing this could be to reduce the number of pre-emption over system's task set. Generally, one of the weaknesses of Rate Monotonic algorithm comes from the high level of overhead that results due to high pre-emptions. Therefore, reducing unnecessary pre-emptions could have a significant impact on algorithm performance, in the same time by keeping its simplicity [7].

4 Reducing the number of pre-emptions

The idea of the proposed method is based on the observations derived from Rate Monotonic Algorithm usage for scheduling real-time tasks (tasks with deadlines), from which a high number of pre-emptions were noticed. As it is well known, every pre-emption induce a run-time overhead and we consider that, by reducing the number of pre-emptions in the resulting scheduling scheme, the overall runtime overhead decreases, and this could result in an increasing efficiency when we speak about real-time applications.In order to avoid pre-emptions, it is important to know when they occur: generally, they take place when a higher priority task is activated during the execution of a lower priority task. Lower priority tasks would experience more pre-emption than high priority ones, as they stay longer in the ready queue.

To reduce the chance for pre-emption, one possible method implies to reduce the period of time while a task stays into the ready queue. It should be possible to do this, and one method is to delay the activation of the task, by setting R_i to a value > 0; of course, this delay should not have implications to the overall schedulability of the task set, and, consequently, must be done under strict control. In order to derive such a method, based on an algorithm that delays start time for a set of tasks, the following task model is considered: every task is denoted by ζ_i; each task is periodic and the method applies itself only for periodic tasks, the period of task ζ_i is denoted by T_i; C_i represents the worst case execution time (WCET) for task ζ_i; P_i represents the priority of task and the priority of each task is fixed; the ratio C_i/T_i represents the utilization

factor of the task ζ_i and represents the fraction of processor time that is used by task ζ_i; the deadline of a task D_i represents a typical task constraint in real-time systems and represents the time before which the task must complete its execution - usually, the deadline of the task is relative, meaning that, from the moment when a task ζ_i arrives, it should finish within D_i time units and in particular, when using Rate Monotonic algorithm, the deadline is considered equal with task period: $D_i = T_i$; task ready (arrival) time denoted by R_i which represents the moment of time when the task ζ_i is ready for execution.

Consequently, the algorithm for ready time modification calculates first the maximum delay time for each task, in order not to affect the task deadline. For task ζ_i that is characterized by a period (and deadline) T_i and has a worst case execution time C_i, the following relation is used:

$$\max\_\text{delay}(\zeta_i) = T_i - C_i. \tag{5}$$

Let's denote with S a task set of size n and consider that tasks are ordered by their priority (as they appear in the ready queue):

$$S = \{\zeta_1\ \zeta_2\ .\ .\ .\ \zeta_{n-1}\ \zeta_n\} \tag{6}$$

where ζ_1 has the highest priority and ζ_n the lower one.

The algorithm picks every task from S, in the decreasing order of their priority, and verifies if it is possible to change the ready time R_i; in order to reduce the possibility of pre-emption for the higher priority task (ζ_i), the algorithm starts by delaying it as much as possible [1]. Verification is based on the maximum delay that is possible for each task, calculated as presented in (5) and take into consideration the computation time for tasks that were already delayed in order not to compromise the schedulability for the given task. The algorithm tries to delay as much as possible tasks with high priority, given to the tasks with low priority the chance to be less pre-empted. The set of tasks that have been delayed during execution of algorithm are included in DELAYED (corresponding to the waiting queue) and corresponding delay is calculated (R_i). This will be used further into the simulation tool for modifying ready times. Consequently, we obtained the following structure of the algorithm:

$R_1 = max\_\,delay(\zeta_1) = T_1 - C_1$;
include (ζ_1) *in DELAYED ;*
for$(i = 2;\ i \leq n;\ i\ ++)$
{
  if $((max\_\,delay(\zeta_i) - \Sigma_{j\ in\ DELAYED}\ C_j) > C_i)$
  {
    $R_i = max\_\,delay(\zeta_i) - \Sigma_{j\ in\ DELAYED}\ C_j$;
    include(ζ_j) *in DELAYED*;
  }
  else
  {
  $R_i = max\_\,delay(\zeta_i)$;
  }
}

5 Case study analysis and results

The idea of the above algorithm is illustrated considering the case study shown in Figure 1. Each task is represented by its computation time C, period T and deadline D, and it is assumed

that release time R for all tasks is null. So, we considered a task set consisting of three tasks ζ_1, ζ_2, and ζ_3 with the following characteristics:

$C_1 = 10;\ T_1 = 30;\ C_2 = 30;\ T_2 = 90;\ C_3 = 20;\ T_3 = 120.$

We simulated the execution of these tasks using the framework developed in [18, 19] and the results obtained are presented in Figure 1. The simulation is carried out using a developed framework which allows modelling, simulation and schedulability analysis for a set of real-time systems tasks. The tool has a graphical user interface for introducing tasks parameters, such as: deadline, execution time, priority (as presented in the left panel) permitting that all these parameters to be saved in a (text) file for each given task, in a specific format. Also, several scheduling algorithms are implemented into the tool, and the user could choose from a list of implemented algorithms. These algorithms are grouped into two categories: for periodic and non-periodic tasks.

For our case study, we consider to work only with Rate Monotonic. The framework uses a built-in simulator that illustrates a graphical representation of the generated trace of execution for the set of tasks, according to the chosen scheduling algorithm (as presented in the right panel). From these results we noticed that task ζ_1 exhibits no pre-emption, ζ_2 exhibits maximum one pre-

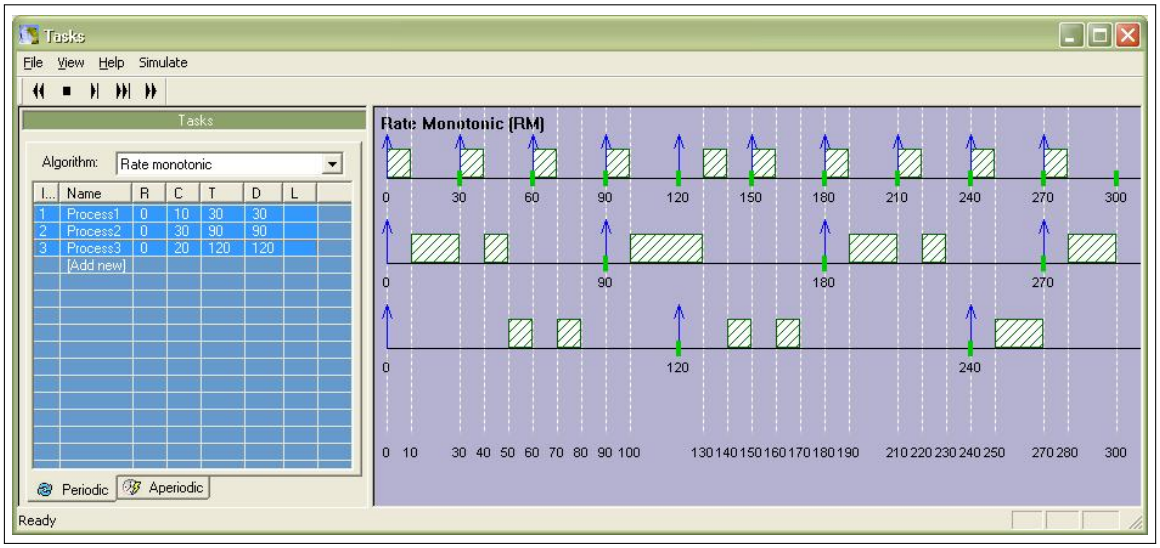

Figure 1: Set of 3 tasks scheduled with Rate Monotonic without modifying start (ready) times

emption and task ζ_3 maximum 2 pre-emptions. By applying start times modifications according to the proposed algorithm, the following release times for the considered tasks were calculated:

$i = 1;$
$R_1 = T_1 - C_1 = 30 - 10 = 20;$
$S = \{\zeta_1\};$
$i = 2;\ \max\_delay(\zeta_2) = T_2 - C_2 = 90 - 30 = 60;$
$60 - 20 = 40 \Rightarrow R_2 = 40;$
$S = \{\zeta_1\ \zeta_2\};$
$i = 3;\ \max\_delay(\zeta_3) = T_3 - C_3 = 120 - 20 = 100;$
$100 - 40 = 60 => R_3 = 60;$
$S = \{\zeta_1\ \zeta_2\ \zeta_3\};$

We modified the release times for our tasks accordingly; consequently, as it is shown into the new simulation presented in Figure 2, we observed that the number of pre-emption for task ζ_3 is

reduced to maximum one. It is obvious that the number of pre-emptions for task ζ_3 is reduced by this modification.

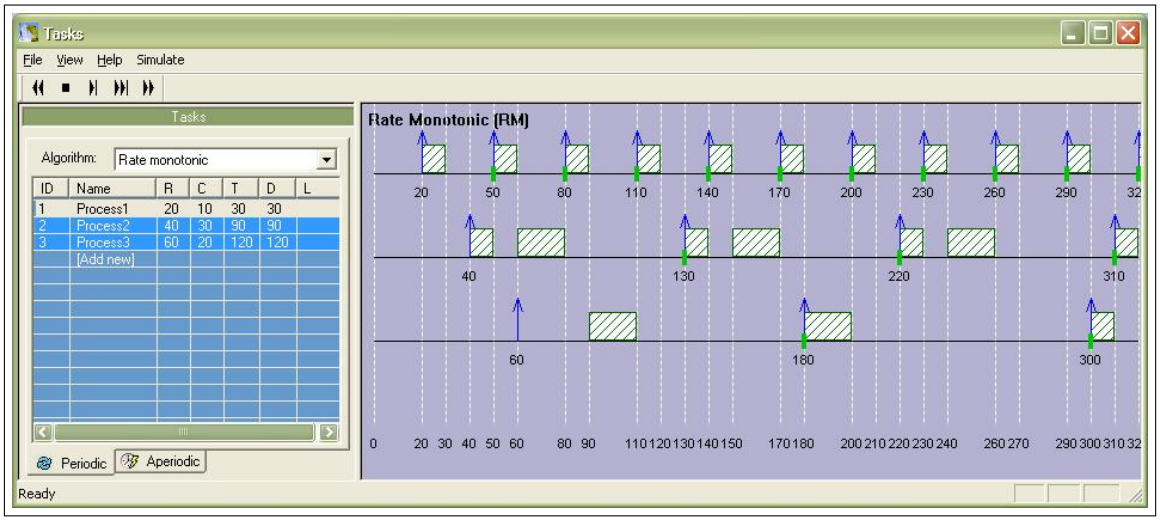

Figure 2: Set of 3 tasks scheduled with Rate Monotonic with modifying start (ready) times

One thing about our resulting model is no concluding is the delay of the least priority task: based on the algorithm idea, it should not be delayed at all, because so it has a chance to run when other tasks have not, due to their delays, and, therefore, the chance for pre-empting it decreases in these conditions. But, in many cases, an overall lower pre-emption rate is achieved.

6 Conclusions

Starting from the premises that Rate Monotonic algorithm is simpler to implement and exhibits a predictable behaviour resulted from its associated analysis, in this paper, a possible adaptation of Rate Monotonic algorithm is proposed, in order to overcome some of its disadvantages when using it in real-time applications.

One aspect that has impact on the model overall accuracy takes into consideration the overheads implied by the scheduling process by including them into the task's computation times, and a possible way of considering this is proposed. Another aspect, with impact on scheduling overall performance focuses on reducing the number of pre-emptions, accrediting the idea that by doing this, the performance of the algorithm increases. Pre-emptions were reduced based on tasks start times modifications, and an algorithm that controls these start time adjustments was proposed.

The algorithm was tested using several use cases (one example presented in the paper), pseudo-randomly generated and the same conclusion was reached: by controlling tasks release times according to the proposed algorithm, in most cases the number of pre-emptions decreases. Thus, the proposed algorithm together with the implemented tool provides a very powerful analysis framework that can be used in real-time application modelling and further development.

Bibliography

[1] A. Aravind and J. Chelladurai, Activation Adjusted Scheduling Algorithms for Real-Time Systems, *Advances in Systems, Computing Sciences and Software Engineering*, pp. 425-432,

Springer 2006.

[2] N. Audsley, On priority assignment in fixed priority scheduling, Fuzzy Control Rules in Convex Optimization, *Inf. Process. Lett.*, 79(1), pp.39-44, 2001.

[3] N. Audsey, A. Burns, R. Davis, K. Tindell, A. Wellings, *Fixed Priority Preemptive Scheduling: An Historical Perspective, Real Time Systems,* vol. 8, pp. 173-198, 1995.

[4] R.J. Bril, P.J.L. Cuijpers, *Analysis of hierarchical fixed-priority pre-emptive scheduling revisited,* TU/e CS-Report 06-36, 2006.

[5] G. C. Butazzo, Rate Monotonic vs. EDF: Judgement Day, *Real-Time Systems,* 2005.

[6] R.I. Davis, A. Burns, Hierarchical Fixed Priority Pre-Emptive Scheduling, *Proceedings of the 26th IEEE Real Time System Symposium,* IEEE Computer Society, pp. 389-398,2005.

[7] R. Dobrin and G. Fohler, Reducing the Number of Preemptions in Fixed Priority Scheduling, *Proceedings of Euromicro Conference on Real Time Systems,* pp. 144-152, 2004.

[8] J. Goossens, Scheduling of Offset Free Systems, *Real-Time Systems,* 24(2), pp. 239-258, 2003.

[9] J. Goossens, R. Devillers, The no-optimality fo the monotonic priority assignments for hard real-time systems, *Real-Time Systems,* 13(2), pp. 107-126, 1997.

[10] C. Kirch, Principles of Real-Time Programming, *EMSOFT02, LNCS 2491,* Springer-Verlag Berlin, 2002.

[11] J. Kollar, J. Poruban, P. Vaclavik, Evolutionary Nature of Crosscutting Modularity, *Proceedings of the 9th International Conference of Modern Electric Systems,* EMES'07, pp. 43 - 48, 2007.

[12] C. L. Liu and J. W. Layland, Scheduling Algorithms for Multiprogramming in a Hard real Time Environment, *Journal of the ACM,* vol. 20(1), pp. 46-61, 1973.

[13] C. L. Liu, *Real-Time Systems*, Prentice Hall, 2000.

[14] M. Naghibzadeh and K. H. Kim, A modified Version of Rate Monotonic Scheduling Algorithm and its Eficiency Assessment, *Proceedings of the Seventh IEEE Internation Workshop on Object Oriented Real Time Dependent Systems,* pp. 289-294, 2002.

[15] I.Shin, I. Lee, Periodic resource model for compositional real-time guarantees , *Proceedings of 24th IEEE Real Time System Symposium,* (RTSS), pp.2-13, 2003.

[16] S. Saewong, R. Rajkumar, J.P. Lohoczky, M.H. Klein, Analysis of Hierarchical Fixed-Priority Scheduling, *Proceedings of 14th Euromicro Conference on Real Time Systems,* (ECRTS), pp. 152-160, 2002.

[17] K. Somasundaram; S. Radhakrishnan, Task Resource Allocation in Grid using Swift Scheduler, *International Journal of Computer, Communication and Control,* ISSN 1841-9836, E-ISSN 1841-9844, vol. IV, no.2, pp. 158-166, 2009.

[18] D. Zmaranda, G. Gabor, Tool for Modeling and Simulation of Real-Time Systems Behavior, *Proceedings of the 2nd IEEE International Workshop on Soft Computing Applications,* SOFA 2007, Gyula, Hungary - Oradea, Romania, ISBN: 978-1-4244-1608-0, pp. 211-215, 2007.

[19] D. Zmaranda, C. Rusu and M. Gligor, A Framework for Modeling and Evaluating Timing Behaviour for Real-Time Systems, *Proceedings of the International Symposium on Systems Theory - Software Engineering,* SINTES vol III, pp. 514-520, ISBN 973-742-148-5, 2005.

[20] C. Gyorodi, R. Gyorodi, M. Dersidan, L. Bandici, Applying a pattern length constraint on the FP-Growth algorithm, Proceedings of the International Workshop on Soft Computing Applications SOFA 2009, IEEE - Computational Intelligent Society, 29 July - 1 August 2009, Szeged-Hungary, Arad - Romania, IEEE Catalog number CFP0928D-PRT, ISBN 987-1-4244-5054-1, pp. 181-185, 2009